Amir K.C.

Trusted Hosts in Host Identity Protocol (HIP)

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Final Year Project Bachelor's Thesis
22 May 2012

Metropolia

Abstract

Author Title Number of Pages Date	Amir KC Trusted Hosts in Host Identity Protocol (HIP) 51 pages + 6 appendices 22 May 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Data Networks and Telecommunication
Instructor Supervisor	Dr. Göran Pulkkis, Project Manager Matti Puska, Principal Lecturer

The goal of the project was to study the possibilities to establish trusted hosts in Host Identity Protocol (HIP) and implement certificate handling in HIP packets. The time complexity and performance while using certificates in HIP packets was also measured. The research project was carried out at Arcada University of Applied Sciences in collaboration with Helsinki University.

The project aimed to implement standard x.509 certification of the public key used as HI (Host Identity) to derive trust between peers. For this purpose, an open source implementation of HIP, OpenHIP v0.7, developed by Boeing Company was modified and extended so that standard x.509 certificate can be encapsulated in HIP packets in order to establish trusted network hosts.

The extension of OpenHIP to support certificate handling in HIP packets was effectively implemented and tested. The results indicate that x.509 certificates can be successfully deployed to derive a trusted relationship between the network hosts. Furthermore, the performance measurement results show that certificate handling does not consume much time for HIP Base Exchange.

Further development of the project can be done to support mobility of the hosts in trusted relationships. Likewise, the current modified version of OpenHIP only supports Debian distributions of Linux, extensions can be made to support other Linux distributions, OSX and Windows platform as well.

Keywords	x.509 certificate, HIP, rendezvous, public key, OpenHIP, Base Exchange, Host Identity, LSI, PKI

Preface

This thesis is the outcome of a research project based on Host Identity Protocol (HIP) which was carried out at Arcada University of Applied Sciences in collaboration with Helsinki University.

I would like to express my hearty gratitude to Dr. Göran Pulkkis, the project manager and senior lecturer as well as my thesis instructor from Arcada UAS for providing me the opportunity to work in the project. Similarly continuous motivation and support from Göran Pulkkis, Timo Karvi and Harri Forsgren from Helsinki University were also much appreciated.

I cordially want to thanks Mr. Matti Puska from Metropolia University of Applied Sciences for accepting my request to supervise my thesis and reviewing it even in his busiest hour. His valuable comments and guidance were quite insightful.

Likewise, I am equally grateful to Ulla Paatola for her kind cooperation and Salla Petäjä, my language advisor for revising my thesis though in critical situation. Last but not the least I would like to thank my lecturers, my family, my girlfriend for their support and encouragement.

Contents

1 Introduction

The advent of Internet has been a remarkable milestone in the history of communication since it has become an essential part of life for millions of people. In the world of internetworking, TCP/IP (Transmission Control Protocol/Internet Protocol) is a fundamental element of communication networks which serves the end-nodes with connectivity. However TCP/IP lacks packet's security, host's mobility and multihoming. To address these issues, Host Identity Protocol (HIP) gets evolved with an approach towards secure and seamless mobile communications through the separation of location and host identity of a host. In the current Internet, IP address serves both as a topological locator and as a host identity for a host.

HIP is designed to be resistant against various attacks like, replay attack, Denial-of-Service (DoS) attacks and Man-in-the-Middle attacks with the use of public key. It however lacks trust in those public keys and in signatures between the peers. Nevertheless, an approach was made to study and implement potential alternatives to derive the trust in public keys. A research project was carried out at Arcada University of Applied Sciences, where I did my work placement and the project was done in collaboration with Helsinki University. The team members include two senior lecturers, Göran Pulkkis and Kaj Grahn, from Arcada University of Applied Sciences, one senior professor, Timo Karvi, and a doctorate student, Harri Forsgren, from Helsinki University and me. The programming work was done by Harri Forsgren whereas my contribution for the project was to build the infrastructure, to test the developed HIP application, documentation and to make measurements on it.

The main goal of the project is to study the possibilities to establish trusted hosts in Host Identity Protocol and implement certificate handling in HIP packets. The project aims to implement standard x.509 certification of the public key used as HI (Host Identity) to derive the trust between peers in the OpenHIP application. Additionally, the project intends to measure time complexity and performance of HIP Base Exchange when certificates are used in HIP packets.

2 Theoretical Background

The Host Identity Protocol is a security protocol which gives cryptographic identities to the host other than IP addresses. It establishes end-to-end secure IPSec Encapsulated Security Payload (ESP) associations and means to provide mobility and multihoming. The general overview of HIP architecture and features are discussed in this section.

2.1 Host Identity Protocol

The credit for introducing the Host Identity Protocol (HIP) goes to Robert Moskowitz from a software development and consultancy firm called ICSA Inc. in Virginia,US. He prepared a first draft draft-moskowitz-hip-00 in the IETF (Internet Engineering Task Force) in May 1999. The draft includes the basic architecture of HIP. Later a number of people in Ericsson Nomadic Lab, Boeing and Helsinki Institute for Information Technology (HIIT) worked together for improvement and modifications of Host Identity Protocol drafts and protocol specification. At first the group was led by Robert himself but later Pekka Nikander led the group. They developed the packet structure for HIP, the state machine and protocol details and published the specification. On 2004 an IETF working group was formed to enhance and define the minimal elements for HIP experimentation in large scale. On this process the working group proposed several drafts and specifications on HIP Base Exchange, Encapsulating Security Payload (ESP), rendezvous extensions, multihoming and mobility. After a period of background development, the first standardized RFC 4423 on Host Identity Protocol (HIP) architecture was published on May 2006 by IETF. [1,8.] The IEFT later on published RFCs on HIP Base Exchange, ESP, registration extensions, rendezvous and Domain Name System (DNS) extension, multihoming and mobility and NAT (Network Address Translation) and firewall traversal respectively on 2008. Several researches on prospective implementation and development for HIP version 2.0 are still going on in different research institutions.

The intent of developing the Host Identity Protocol is to separate the host identifier and location identifier. Currently both host topological identifier and location identifier are defined by the IP address of the host. When a host changes the location, it must change the IP address to be reachable. In order to get out from depending on the IP

address of a host for both host and location identity, the HIP protocol brings a new mechanism of separating host and location identifier from the IP address with the introduction of new layer in the TCP/IP stack called Host Identity layer. The new layer added between the network and transport layers, maps the host identifiers to the different locations of the host. When decoupled, the IP address will continue acting as location identity while HIP will be responsible for serving host identity. [2,11.] Figure 1 shows the separation of locator and identity of a host.

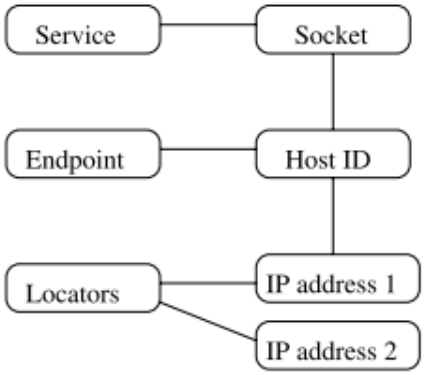

Figure 1. Separation of locator and identity of a host. (Extracted from [1,5])

As shown in figure 1, IP addresses of the host serve as the locators whereas Host Identity serves as the identity of the host. . The HIP consists of mainly three parts; HIP Base Exchange, ESP (Encapsulated Security Payload) payload and rekeying of the Host Identity for mobility and multihoming. This allows the easy implementation for mobility of the hosts. Hence, HIP can be regarded as multi-addressing as well as a mobility solution for the Internet. [3,1.]

2.2 Architecture

HIP introduces a new layer in the TCP/IP protocol stack. The new layer added between the network and transport layers, maps the host identifiers to the locations of the host and vice versa [3,1]. It separates the host's identifier from the IP address. Usually IP address serves both the host's locator and identifier. In HIP architecture IP addresses act as locators whereas Host Identifier (HI) acts as an end-point identifier. Figure 2 shows the new protocol stack of Host Identity Protocol.

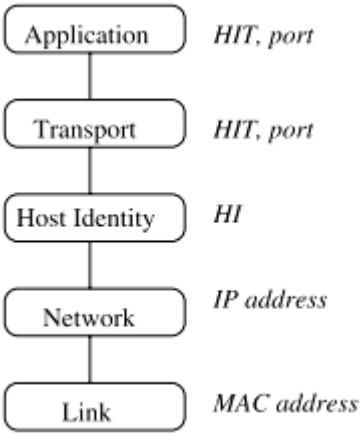

Figure 2. The protocol stack of HIP. (Copied from [1,8])

The new host identity layer is inserted between the network and transport layers as shown in figure 2. A physical computer host could have several logical interfaces and each interface will have a distinct Host Identity [2,12]. If a client host moves from one HIP supported DHCP (Dynamic Host Configuration Protocol) enabled network to another, the client needs not to do anything and maintain the communication without disconnecting in case of ongoing communication. The new IP address will be communicated to source host using HIP.

2.3 HIP Packet Format

All HIP packets start with a fixed header [4,31]. Figure 3 shows the format of a HIP packet which carries HIP related information from one node to another.

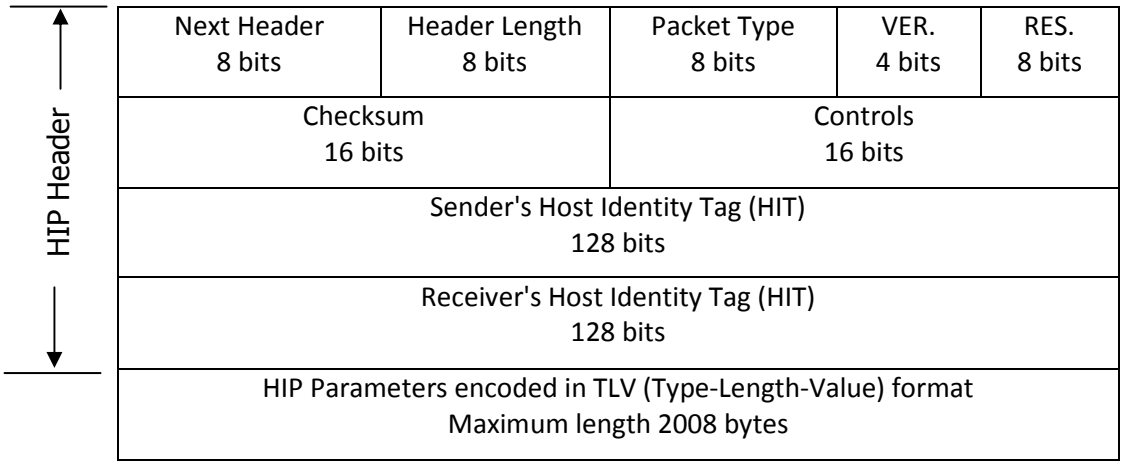

Figure 3. HIP packet format. (Adapted from [4,31])

As depicted by figure 3 a HIP packet consists of fixed header size and a HIP parameter field with variable length. The current implementations of HIP do not use the Next Header field. However, the value of the next header is decimal 59 currently which means there is no Next Header. [4,32.]

The Header Length field shows the total length of the HIP Header and HIP parameters in 8-byte units but it excludes the first 8 bytes. It includes length of Sender's HIT, Receiver's Host Identity Tag (HIT) and HIP Parameters field. It includes 16 bytes of Sender's HIT, 16 bytes of Receiver's HIT and length of HIP parameter field if the field has been used. The Packet Type field indicates the type of the HIP packet. HIP Packet Type 1 indicates I1 packet, Type 2 indicates R1, Type 3 indicates I2 and Type 4 indicates R2 packet [5].

The HIP Version field indicates the version of the HIP and the current version is 1. The three bits of Reserved field are reserved for future use. The value for the reserved bits is 0. The Checksum field is 16 bits long that contains the source and destination addresses that must be recomputed on HIP-aware NAT devices. The Controls field is also 16 bits long. It contains information about the packet structure. Only one field of the Controls is used and other fields are reserved for future use. If the last byte of the field is set as A which denotes anonymous HI, it refers that the sender's HI is not listed in the directory. [4,33.]

The Sender's Host Identity Tag (HIT) and Receiver's Host Identity Tag (HIT) field is 128 bits long and as the name suggests it contains the source and destination HIT. The HIP Parameters field contains HIP signaling information associated with security and other information. The parameters are encoded in prescribed Type Length Value (TLV) format and are ordered consecutively. All parameters have a length in a multiple of 8 bytes. The parameters are placed in the HIP packet in increasing order otherwise the packet is dropped considered as a malformed packet. The TLV parameters that are numbered between 0 to 1023 are used in HIP handshake and update procedures whereas parameters numbered between 1024 to 2047 and 4096 to 61439 have been reserved for future use. The parameters numbers between 2048 to 4095 are used in HIP transform types. Similarly the parameters numbered between 61440 to 62463 are

used for signature handling and numbered between 63488 to 64511 are used for the rendezvous mechanism. [4,36.]

2.4 Identifying Hosts

A pair of public and private keys which is generated with Rivest Shamir Adelman (RSA) algorithm by default or Digital Signature Algorithm (DSA) algorithm provides the Host Identity for a host. The length of the public key can be 512, 1024 or 2048 bits. The current implementations of HIP usually generate the public and private keys during installation of HIP software but they can be generated afterwards also. As mentioned above the length of the public key can be variable in size and fitting the public key in HIP packet is problematic because of size as the typical Maximum Transmission Unit (MTU) is 534 bytes. To resolve this problem and maintain the compatibility with Berkeley socket interface two other forms for identifying the host that are derived from Host Identity are introduced. [1,46.] They are Host Identity Tag (HIT) and Local Scope Identifier (LSI).

The Host Identity Tag a one-way SHA-1 (Secure Hash Algorithm) hash of the public key. It has a fixed length of 128 bits regardless of the public key length and cryptographic algorithm used in public–private keys. Figure 4 shows the methods of identifying the host in HIP.

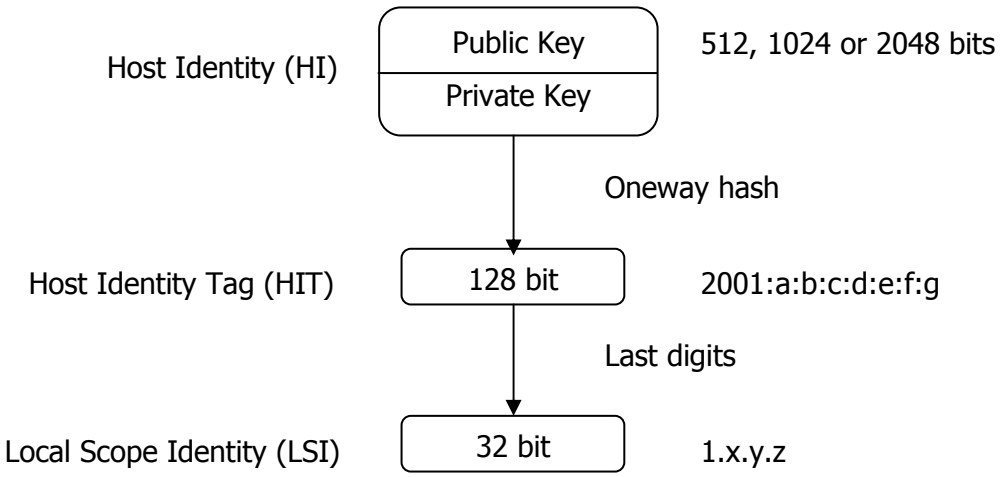

Figure 4. Methods of identifying a host. (Reprinted from [1,46])

Figure 4 shows how the HIT and LSI are derived in Host Identity Protocol. HITs are statistically and globally unique and have a prefix of 2001:0010::/28. Though on viewing the format of a HIT and an IPv6 address look similar, the prefix /28 distinguishes them. [1,46] The Local-Scope Identifier (LSI) is the last 32-bit identifier derived from the HIT. As the name suggests, LSIs have only local significance and cannot be globally unique [1,47]. The LSI has the same length as IPv4 addresses but LSIs have a prefix 1 to distinguish them from publicly allocated IPv4 addresses. The typical format of the LSI is 1.x.y.z.

2.5 Base Exchange

The HIP Base Exchange is a four-way handshake between two hosts. In this process, they exchange security credentials and cryptographic keys to establish the secure HIP session. [6] The host that starts the Base Exchange is called initiator and the other host who sends answers to the initiator is called responder. Figure 5 shows the four-way handshake between the hosts using HIP.

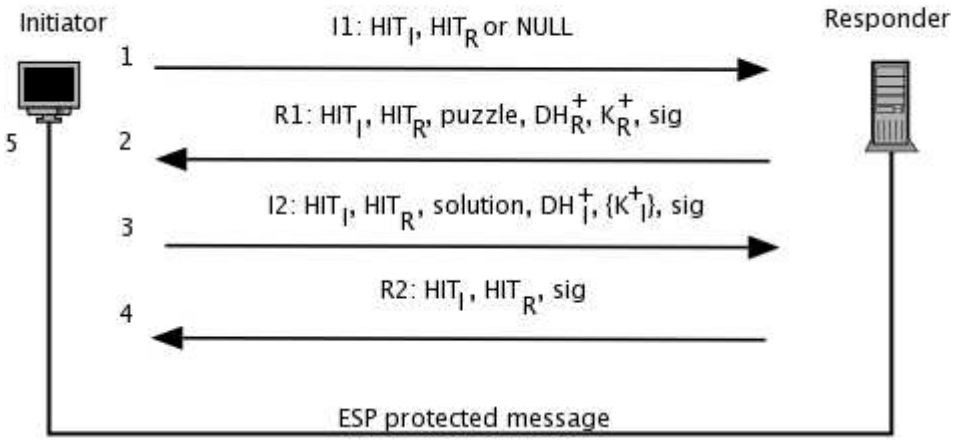

Figure 5. Base Exchange. (Reprinted from [7])

The Base Exchange includes exchange of four HIP control packets as illustrated in figure 5, I1 is the first packet used to trigger the Base Exchange and the three other packets R1, I2, R2, are used to establish the security associations between the initiator and responder based on the solution of a puzzle, verifying it and Diffie-Hellman key exchange.

I1 packet

The first message sent by initiator to trigger the HIP handshake is an I1 packet. It contains the Host Identity Tag (HIT) of the source that is referred to as initiator and, if known, the Host Identity Tag of the destination that is responder. If the responder´s HIT is unknown, the I1 packet has NULL value. [8,16.] In this case, the initiator should start the HIP handshake in the opportunistic mode. Initiator sends the I1 packet to the responder's IP address. When I1 packet is sent to the responder, initiator expects R1 packet as a reply after few milliseconds of time frame. After a timeout, if it does not receive the R1 packet, initiator retransmits I1 packet again [1,52]. If a Rendezvous Server mechanism is used in the network infrastructure, the I1 packet is sent to the Rendezvous Server. The server looks up the IP address of the responder and transmits the I1 packet to the responder.

R1 packet

R1 is the packet that the responder sends to the initiator as a response to the I1 packet. When the application starts, the responder precomputes the partial R1 packet. Several R1 packets are generated when the application is started. The precomputed R1 contains HIT of the responder, the responder's Diffie-Hellman key, HIP transforms containing the proposed cryptographic algorithms, the proposed IPsec algorithms and an Echo_Request field. The data that the initiator has to return unmodified in I2 packet is included in the Echo_Request. In the precomputed R1 HIT of initiator and puzzle field are left empty and as soon as responder gets I1, these fields are populated. [9,482.] When the initiator started the Base Exchange sending the I1 packet, the responder selects one of the R1 packets that were precomputed and sends to the initiator. Hence, the R1 packet contains a Diffie-Hellman parameter to create the session key for the establishment of security associations, a cryptographic puzzle that is to be solved by the initiator and has to be sent in I2 packet for proceeding the Base Exchange and the responder's Host Identity (HI). The cryptographic puzzle contains a random number and a defined difficulty in the configuration file [8,17]. The default difficulty level is 10. The puzzle mechanism is included by the responder to avoid

Denial-of-Service (DoS) attacks from the fake hosts. The responder uses its private key to sign the packet.

I2 packet

I2 is the second packet from the initiator and is a response to the R1 packet. When the initiator receives an R1 packet from the responder, the initiator first verifies the signature using the public key of responder. If the signature is verified, the initiator solves the cryptographic puzzle sent by responder. The puzzle has three components: a random number I, the difficulty level K, and the solution J. For the value of J, the initiator carries out a brute-force computation several times. When the matching value is found, the initiator creates the I2 packet including the solution and sends it to the responder. Along with the puzzle solution, I2 includes initiator's Diffie-Hellman key, the HIP and ESP transforms proposed by the initiator, security parameter index (SPI) for the IPsec security association (SA), and the Echo_Response. [3,3.] A Hashed Message Authentication Code (HMAC) is included in the packet for an additional protection against attacks. The packet is signed before the transmission.

R2 packet

R2 is the second packet from the responder and is a response to the I2 packet. It R2 is the fourth and the last packet in HIP Base Exchange. If the puzzle is solved and all parameters are correct, a HIP connection is established and data can flow between the two hosts. On receiving I2, the responder verifies the puzzle solution. If it is correct, the responder computes the session keys, decrypts HI-I, and verifies the signature on I2. The responder then sends R2, which contains the SPI for the initiator-to-responder IPsec SA, an HMAC computed using the session key and a signature. For the initiator, the exchange is concluded by the receipt of R2 and the verification of the HMAC and the signature. The HMAC confirms the establishment of the session key for security association. [3,3.] In the responder side, the key confirmation is made by the first IPsec packet that is protected with the new security association.

2.6 Mobility and Multihoming

Once the Base Exchange is completed, the security association does not depend on IP addresses. The host is able to receive packets from any address with the HIP protected ESP Security Association (SA). So the host can move and change its IP address without disconnecting with the peers and this feature is called mobility. Figure 6 depicts the mobility process.

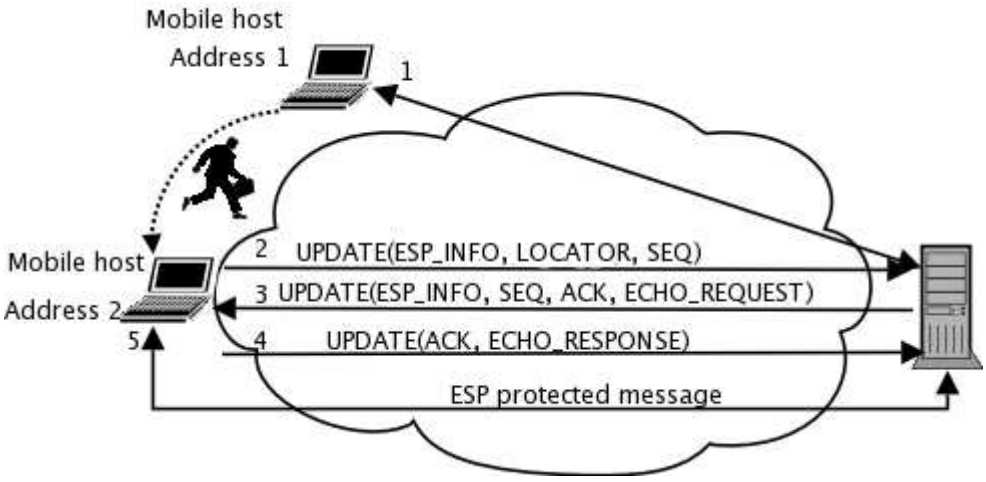

Figure 6. Mobility service in HIP. (Reprinted from [10])

As shown in figure 6, the host was at address 1 and it moves to the address 2 later. While moving the mobile host gets disconnected for a very short period of time and at that time it switches from address 1 to address 2. When a new IP address is gained, the mobile host sends the new IP address to the peer host in an UPDATE message and the update is made with readdressing and optional rekeying followed by the continuation of connection. [11] Likewise, a host can have multiple interfaces. For instance the host has two IP addresses. In this case, the host can notify the peer host with the UPDATE message about the additional interfaces and choose the preferred address. The peer host updates the information and sends UPDATE packets to each IP address including corresponding SPIs (Security Parameter Index). In this way, HIP supports both mobility and multi-homing.

2.7 Rendezvous Mechanism

A mobile host moves and changes its IP address frequently. In this case the other host cannot start the HIP association with the mobile host since the IP address of the mobile host is unknown. The Host Identity Protocol (HIP) architecture introduces the rendezvous mechanism to overcome the limitation of this situation and to help a HIP node to contact a frequently moving HIP node. In the rendezvous mechanism, a Rendezvous Server (RVS) is added, which serves as an initial contact point when initiating the HIP Base Exchange. [12,3.] The Rendezvous Server (RVS) acts as a HIP registrar. To receive rendezvous service, the rendezvous client needs to register by using the HIP Registration Extension. When the client is successfully registered, Rendezvous Server starts to relay HIP packets to a Rendezvous client. After this registration, the initiator starts the Base Exchange using the IP address of the RVS instead of the IP address of the responder when they attempt to establish a Security Association (SA). [13,29.] The Rendezvous Server then relays I1 control packet to the responder and other HIP control packets are transmitted directly to and from the responder and the initiator. Figure 7 shows the procedure of HIP Base Exchange through the RVS.

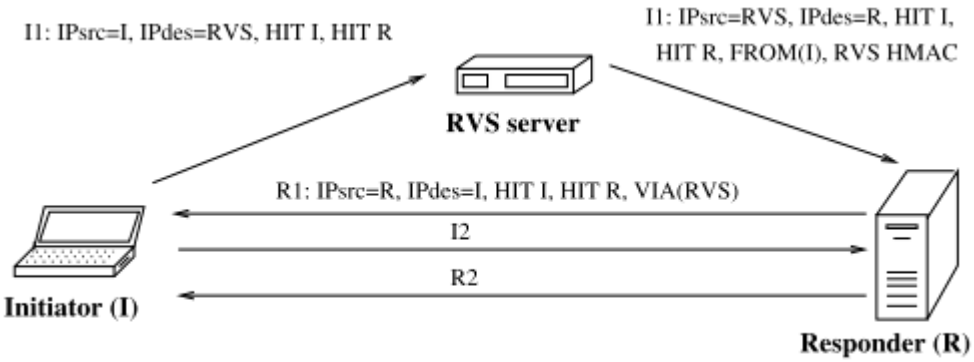

Figure 7. HIP Base Exchange using RVS mechanism. (Copied from [1,78])

As shown in figure 7 the first HIP packet, I1 is sent to responder through the RVS while other packets are transferred directly from the initiator to responder and vice versa.

2.9 DNS extension

The HIP DNS (Domain Name System) extension provides the system for use of HIP with human-friendly host names. A DNS server stores the resource record (RR) data including host public key, HIT and the domain name of a rendezvous server. Figure 8 illustrates the steps of Base Exchange using DNS server.

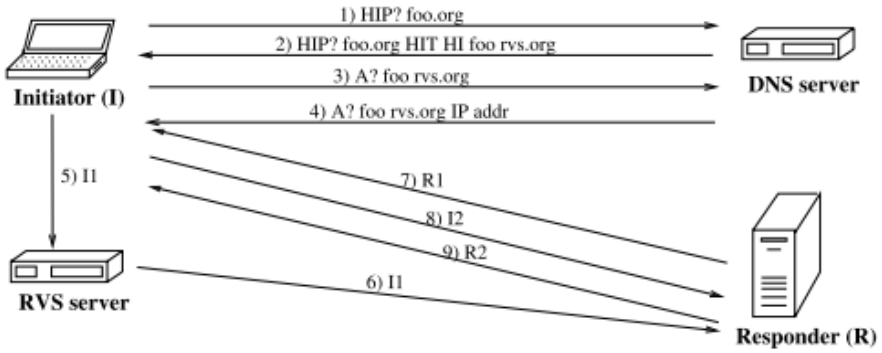

Figure 8. DNS query for HIP mobile host. (Copied from [1,81])

As figure 8 shows, firstly a DNS query is made from the initiator to fetch the information of RVS server. Like IPv6 data HITs are stored in the form of AAAA records in a HIP-aware DNS server. Then RVS server passes the I1 packet to the responder [14,10]. Upon getting replied with R1 packet, other packets travel directly and the Base Exchange is completed exchanging the session key with each other. The method of storing HIs in the DNS server prevents Man-In-The-Middle attacks.

3 Security Mechanisms in HIP

HIP provides high-level of security, confidentiality and authentication for payload and packets. It protects the hosts from various Denial-of-Service (DoS) and Man-in-the-Middle (MitM) attacks.

3.1 Puzzle Mechanism

The purpose of the HIP puzzle mechanism is to protect responders against Denial-of-Service attacks. It delays the Base Exchange by enforcing initiator to solve the puzzle prior to other proceedings. The puzzle solution verifies that it has been generated by the responder and solved by the initiator which checks the sincerity of the initiator. The puzzle calculation is based on the initiators HIT. By so doing, the responder can later verify from I2 that the puzzle has been solved using the initiator's HIT. [4,12.] The responder can make the puzzle difficult for the initiator. Depending on the level of trust of initiator as well as the load of the responder the difficulty of puzzle can be set.

3.2 Security Associations

Security Associations (SAs) are created for secure data communication between the nodes. The purpose of SAs is to establish a pair of IPsec (Internet Protocol Security) ESP between the hosts. A Security Association is identified by three parameters; security parameter index (SPI), destination IP Address and a security protocol identifier. As discussed in mobility section, once the Base Exchange is completed the node can change its topological location and continuously send and receive packets to and from its peers. In the mobility procedures, the origin of HIP control packet is verified by the packet signature. [15,20-21.] To implement the IPSec ESP support in HIP, Bound End-to-End Tunnel (BEET) mode is used. The BEET mode is a combination of transport and tunnel IPsec modes which creates a tunnel between two end hosts. The BEET mode separates an IP address into two parts: inner and outer address. The outer address is IP address itself whereas the inner address is a HIT which is fixed for the lifetime of the SA. The outer address can change from time to time during the SA lifetime without breaking the SA. [1,64-65.]

3.3 HIP Replay Protection

The HIP protocol also incorporates protection against malicious replays. Presigned R1 messages and the puzzle mechanism provide protection against replays of I1 packets. The optional use of opaque data protects against false I2 packets. Protection against R1 replays is done by increasing the R1 generation counter and the counter is kept across system reboots. The HMAC verification takes care of protection against replays of R2 packets and UPDATE packets. The peer host uses Diffie-Hellman session keys to verify the HMAC. [4,14.]

4 Trust In HIP

The current implementations of HIP use RSA and DSA algorithm for signature. However, Identity based Encryption (IBE) is another alternative that can be implemented. Upon receiving the HIP packet from the peer node both initiator and responder verify the signature with the public key. The verified signature is trustworthy only if the receiver is completely convinced that the public key is the HI of the peer. The original HIP specification lacks this aspect since nothing is proposed for gaining the trust between the hosts. Nonetheless, the use of certificates has been defined in the draft presented by Samu Varjonen and Tobias Heer in 2011 [16]. For establishing the trust relationships between the network hosts, mainly two methods are discussed in this thesis report. The trust in a HI can be derived either from the signature of a DNSSEC (Domain Name System Security) response to a DNS query about the IP address, HI, and HIT for a network host named by a Fully Qualified Domain Name (FQDN) or from the use of certificates along with or without a Public Key Infrastructure (PKI) [17,3.]. For the use of certificates there are three alternatives and they are:

- Standard X.509.v3 certification of the public key used as HI
- Identity based public keys
- Certificate-less public keys

For the identity based public keys, any unique identity string can be used as public key. For instance, a fully qualified domain name or a phone number or an email address can be a public key. The private key is generated by a Private Key Generator (PKG). The trust in HI depends on the authenticity of the PKG and its public parameters. [17,4.]

The certificate-less public keys a modification of Identity-based public keys. A network host generates its public key used as HI. A hash of an identity string serves as a partial public key of a HIP host. In this alternative the corresponding private key is created partially by a PKG and partially by the network host. The HIP host chooses a secret value that is embedded in the private and public keys derived from the partial keys. [17,6.]

4.1 Domain Name System Security Extensions

An initiator has to make a DNS query to get the responder's IP address, HI and HIT of responder for the completion of Base Exchange. The DNS query returns the IP address, HI and HIT of responder. In case rendezvous mechanism is used, IP address of RVS server is returned instead of IP address of responder whereby the responder's IP address can be achieved. If the initiator uses an ordinary DNS, there could be crucial condition that an attacker can forge IP addresses and/or HIs and HITs. As a result, the attacker can start a Denial-of-Service attack. Basically, a HIP responder is prepared against this kind of Denial-of-Service attacks, but if there are many of these simple attacks, it can be troublesome. In addition, if HI and HIT are also false, the attacker can impersonate and try to fool the initiator to start communication with the attacker. This sort of impersonation attacks can be detected and avoided only if the initiator has the public key parameters or certificates of the intended responder. In order to avoid these kinds of forgery problems, the initiator can use Domain Name System Security (DNSSEC). A DNSSEC query returns the IP address, HI and HIT of the responder and the packet is signed by the DNS server so that the initiator can trust the data it receives. [18,4.] Additionally to prevent spoofing attacks on DNS resolver queries, the use of DNSSEC is also strongly recommended.

The trust derived from a DNSSEC signature is trust of the initiator in the HI and HIT of the intended responder [18,4]. However, no trust of the responder in the HI of the initiator can be retrieved from DNSSEC signatures in a HIP Base Exchange. It is, of course, possible for the responder to send a DNS query about the initiator in order to be sure about the initiator's identity, but this seems to be troublesome. The DSNSEC brings extra computational costs as the initiator must verify the signature of the DNS server it is using. Moreover, every signature verification should contain a revocation list check in order to detect corrupted public keys.

4.2 Public Key Infrastructure

A responder returns its public key parameters in the R1 message which is also signed. The basic signature consists of all fields in the packet including the initiator's parameters. However, if the puzzle mechanism is used, the responder can prepare the

signature already beforehand and in this case the signature does not include the initiator's parameters. The initiator must verify the signature using the public key of the responder. The initiator may have this public key already beforehand, or it can come with the DNS query and with the R1 message for the first time. In the first alternative, the verification of the signature reveals the forgeries done in the DNS query or in the R1 message on the condition that the public key is valid. Thus, it is necessary that the initiator has the certificate of the responder's public key. Then the initiator can check the CA (Certificate Authority) signature of the certificate and also to check the revocation lists. Certification of the HI of an initiator by a trusted CA gives similar protection to a responder against a malicious initiator since the certification check of the HI in an I2 message reveals a fake initiator. [17,4.]

HI certification by a globally trusted CA could create a trust relationship in the Base Exchange between any pair of HIP network nodes. Presently there is no better authorization of such global trust than pre-installed CA certificates in certificate databases of www browsers and email clients. The HI certification by an intra-domain trusted CA can create a trust relation relationship between two HIP network hosts from different network domains, when both domains belong to the same Identity Management Federation based on a framework like Microsoft CardSpace, OpenID, Liberty Alliance (LA), or Shibboleth. [18, 39.]

4.3 Implementation of Certificates

The CERT parameter defines the encapsulation and transmission of certificates in the parameter field of a HIP packet. The CERT parameter can be used in all the HIP packets but use of CERT parameters is not recommended in the I1 packet as it could cause a Denial-of-Service (DoS) attack. Additionally, the processing time for I1 increases and processing large number of I1s with certificates can be problematic. [19,3.] So the CERT parameter can be used in any packets after I1. The R1 packet is the first signed packet from the responder and I2 is the first signed packet of the initiator. The maximum length of the HIP Parameter field is 2008 bytes. Thus two simple certificates can be delivered in one HIP packet. If the certificate chain is longer or the certificates contain more information, the certificates are transmitted in multiple

packets. The HIP Certificates draft defines the following types for the X.509 certificates [16,4]:

- X.509 v3
- Hash and URL of X.509 v3
- LDAP URL of X.509 v3
- Distinguished Name of X.509 v3

If a certificate chain is long, hash and URL (Universal Resource Locator) of a certificate can be used and it could solve the multiple packets problem. The other possibility is to use multiple packets. When dividing a certificate chain into several packets, the packets must contain the fields CERT group and CERT count. [16,4.] Moreover, extra messages and parameters are used to demand certificates or to inform that more certificates are coming. Figure 9 shows the proposed Base Exchange mechanism between initiator and responder for certificate handling.

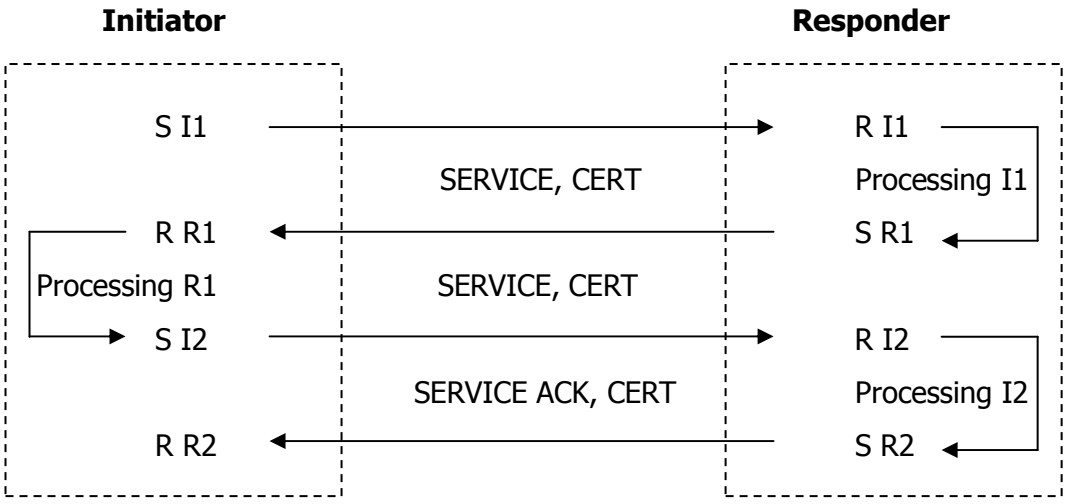

Figure 9. Base Exchange with certificates.

As show in figure 9 the responder asks certificates from initiator by sending the SERVICE parameter in the R1 packet. Initiator responds by expressing that the certificates are coming after the Base Exchange. It is also possible for initiator to demand certificates from responder. In this case initiator sends service offer in the I2 packet. In figure 9, S denotes sent packet and R denoted received packet.

5 Current Implementation of HIP

There has not been lots of implementation of Host Identity Protocol. In this section, the current implementations of the HIP application are discussed.

OpenHIP

OpenHIP is an open source and free software implementation of HIP developed by Boeing Company. The latest release of OpenHIP application is OpenHIP 0.9 released on 23rd of March, 2012. OpenHIP supports both 32-bit and 64-bit version of Linux environment but in Windows environment, it supports only 32-bit version of Windows XP, Windows Vista and Windows 7. Similarly it is also compatiable with OS X of Apple Inc. [20] OpenHIP also supports DNS extensions of HIP and is capable of fetching HIP data from Domain Name Server (DNS). Additionally, the Linux version of OpenHIP has built-in Rendezvous Server too. [21,28.]

InfraHIP

InfraHIP is the project run by Helsinki Institute for Information Technology (HIIT) and Aalto University (formerly Helsinki University of Technology). Host Identity Protocol for Linux (HIPL) is the open source software implementation developed by InfraHIP. HIPL is compatible with several popular Linux distributions. The source and binaries are freely available in their project site. The latest version of HIPL, HIPL Release 1.0.7, was released on 29th April, 2012. HIPL bundle also includes a public-key based firewall module which can be used in middlebox devices such as routers and wireless access points. HIPL needs Linux system with kernel version 2.6.27 or higher which has BEET IPsec support. [22]

Hip4net

HIP for Internet project is developed by Nomadic Labs at Ericsson Research Centre, Finland. Hip4inter is only supported for FreeBSD Linux operating systems. [23] The freely available version of Hip4inter does not support Rendezvous server mechanism but the licensed application with Rendezvous support can be purchased.

PyHIP and CuteHIP

PyHIP is an implementation of HIP using the Python programming language by Andrew McGregor. It was last updated in November 2003 [21,37]. CuteHIP is the Java-based implementation of Host Identity Protocol developed by Dmitriy Kuptsov from HIIT. [24]

6 Extension of OpenHIP

The trust can be established between the network hosts with either use of DNSSEC or use of certificates for public keys representing Host Identities. From the two options, the implementation of certificates as discussed in section 4.3 was chosen to be deployed in this thesis. For this reason instead of developing a new application an open source application, OpenHIP version 0.7, was modified and extended with certificate support. The C language is used as programming language for extending the OpenHIP. Several lines of codes and functions were added in the original source mainly for handling and verifying the certificates in HIP packets. Eventually, the extension of OpenHIP application with certificate support was released and tested. The extended OpenHIP uses x.509v3 certificates for public keys the certificates and verifies certificates. The extended HIP application runs on both 32-bit and 64-bit version of Linux machine. For now the supported operating system is only Debian distributions of Linux. However, Windows compatible version is yet to be developed.

7 Network Infrastructure

The network infrastructure to test the extended OpenHIP is designed in a lab at Arcada University of Applied Sciences. Figure 10 shows the network topology.

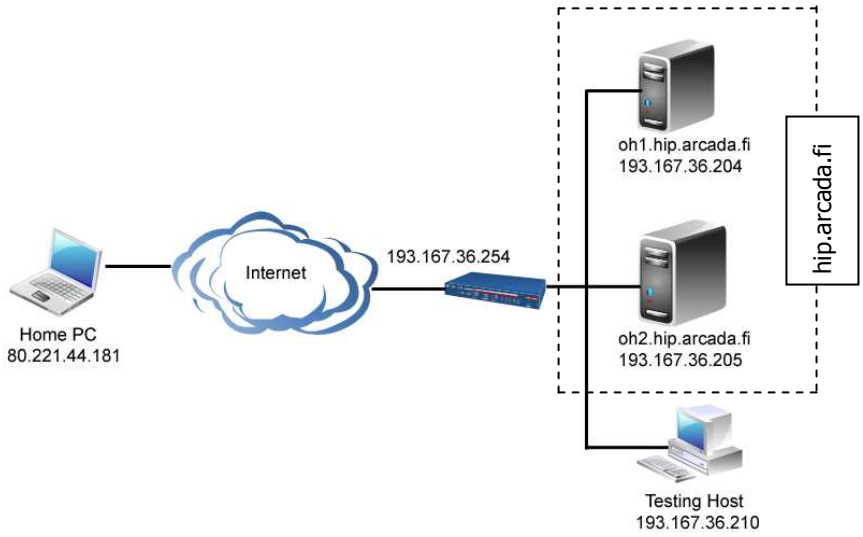

Figure 10. HIP network infrastructure at Arcada University of Applied Sciences.

As shown in the figure 10, two virtual Linux servers hosted in Vmware Server were installed in the domain hip.arcada.fi with corresponding public IP addresses. The original OpenHIP version 0.7 was installed in the host oh1.hip.arcada.fi and the extended OpenHIP was installed in the host oh2.hip.arcada.fi machine. A Linux host and two other public IP address were provided for testing purpose from the local IP network. All hosts in the infrastructure were running Debian distributions of Linux. Table 1 shows the details of the hosts.

Table 1. Network Host details.

1.	oh1.hip.arcada.fi	IP	193.167.36.204
		Gateway	193.167.36.254
		LSI	1.6.65.134
		HIT	2001:1a:3d9c:266:2142:3d3c:ca06:4186
		OS	Debian 2.6.32, 32-bit OS
		Software	original OpenHIP v0.7
2.	oh2.hip.arcada.fi	IP	193.167.36.205
		Gateway	193.167.36.254
		LSI	1.113.244.150
		HIT	2001:12:397f:d41:aeab:6e33:9071:f496
		OS	Debian 2.6.32, 32-bit OS
		Software	modified OpenHIP
3.	Testing host	IP	193.167.36.210
		Gateway	193.167.36.254
		LSI	1.46.253.22
		HIT	2001:1d:754d:50d9:b953:1ca4:d72e:fd16
		OS	Debian 3.1.0, 64 bit OS
		Software	Original OpenHIPv0.7 and Modified OpenHIP
4.	Home PC	IP	80.221.27.14
		Gateway	80.221.40.1
		LSI	1.137.175.153
		HIT	2001:15:4dc9:4d8d:2cd8:e61:8089:af99
		OS	Debian 3.1.0, 64 bit OS
		Software	Original OpenHIPv0.7 and Modified OpenHIP

As illustrated by table 1, four hosts were actively used in this project. Table 1 also shows the respective IP addresses, gateway to connect to Internet, LSI, HIT, operating systems (OS) and installed software in the hosts. For testing purpose 64-bit operating systems were used whereas the server hosts have 32-bit operating systems.

8 Installation of Extended OpenHIP

8.1 Pre-requisites

For the installation of the extended version of OpenHIP, some dependent libraries are required, namely openssl, cryptographic libraries, libxml2, XML libraries, libpbc, PBC-libraries and libgmp, GMP-libraries. The eXtensible Markup Language (XML) library is used for generating and parsing the configuration files. The cryptographic library provides hashing, encryption and public key signing and verification. The Pairing-Based Cryptography (PBC) library performs the mathematical operations underlying pairing-based cryptosystems [25]. The GNU Multiple Precision arithmetic (GMP) library is used for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers [26]. The requirements for the hip daemon can be checked with the following command `ldd ./hip`. The OpenSSL (Secure Socket Layer) libraries and XML libraries were installed with apt-get install openssl and apt-get install libxm2 respectively. The PBC libraries and GMP libraries were not found in the debian repository. So the debian packages are firstly downloaded and installed with the `dpkg -i <package name>` command. The development libraries of GMP version 5.0.1 and development libraries of PBC version 0.5.10 were installed.

8.2 Installation Steps

The HIP application was provided with the source directory in compressed form. The source was then extracted, compiled, built and installed. When building the source the linker searches for the crypto, XML, PBC libraries and GMP libraries. The successful build makes the hip binary. Then `make install` was applied to install the extended OpenHIP in oh2.hip.arcada.fi. The successful installation, by default, installed two identities files known_host_identities.xml and my_host_identities.xml in /usr/local/etc/hip/ directory and a configuration file hip.conf in /usr/local/sbin/. Additionally, the build process installed two executable hip and hitgen in /usr/local/sbin/ directory. [20] Following commands were used to install the application:

```
gunzip hip.tar.gz
tar xzvf hip.tar
cd hip
./configure
make
make install
```

In the same way, the original version of OpenHIP 0.7 was installed in the oh1.hip.arcada.fi host. The pre-requisites for the original OpenHIP exclude the PBC and GMP libraries.

8.3 Configuration

For running the extended openHIP, some changes were made in the configuration file hip.conf and in the two identities files, namely known_host_identities.xml and my_host_identities.xml.

hip.conf

The configuration file, hip.conf, is the only configuration file where different parameters for HIP daemon are defined. The configuration file is written using XML language. In the XML language, different parameters are written inside tags (<>) and the value for the parameter is placed between the opening and ending tags. The final hip.conf file is included in Appendix 1. In addition to the default parameters, few extra parameters were added in the hip.conf file for certificate handling. The hip.conf file is normally generated while installing the application. In case there is no configuration file in the /usr/local/sbin/ directory, it could be generated with the hitgen command. The command hitgen -conf generates the default configuration file and saves it in the default directory.

The hip.conf file consists of several parameters such as cookie_difficulty which sets the puzzle difficulty. The puzzle is sent by the responder to the initiator. The initiator has to solve the puzzle within the specified lifetime to accomplish the Base Exchange. The default value is 10. The cookie_lifetime sets the lifetime for the puzzle to be solved. The default lifetime is 128 seconds. The number of packets to be retransmitted before

indicated failure can be also set in the max_retries option. The other few significant options are dht_server, dht_server_port and dns_server. The dht_server option takes the IP address of the Bamboo DHT (Distributed Hash Table) server if DHT server is used instead of DNS server. DHT server stores the HIP information and maps the IP address to the HIT. By default, the dht_server is not active. In the same way, dht_server_port takes the XML RPC (Remote Procedure Call) port for the DHT server. The dns_server option is used for specifying IP address of the DNS server that stores the HIP RRs. This option is optional and if no server is specified, the system's default DNS server is contacted for HIP RR data. Furthermore, the save_known_identities option determines whether to save or not the learned identities to known_host_identities.xml. [20] The allowed options are either true or false. The default value is true. Some parameters were added in hip.conf for handling the certificates.

All configuration settings are left in the default state except two parameters; disable_dns_lookup and save_known_identities. The default value for both parameters is no. The default settings were changed to yes in both cases to disable the DNS lookup and save the learned identities in the known_host_identities.xml file.

The additional XML parameters that were added in the extended OpenHIP are my_certificate, send_certificate, peer_certificate_required and pkgparams. The my_certificate option specifies the location of the certificate file or certificate chain file. The send_certificate option determines whether to send or not to send the host's certificate. The options for this tag are either yes or no. The peer_certificate_required option specifies whether the peer's certificate is required or not. The option is either yes or no. The pkgparams tells the location of a file containing PKG parameters delivered in service offer parameter. File pkgparamsig.bin is generated in two parts. First the PKG public parameters are generated by hitgen by giving a "-pkgfile pkgparam.bin" parameter. Next a signature for that file is generated with OpenSSL using PKG's secret RSA key. Then, the PKG parameter file and the signature file are concatenated together. Hence, four new configuration XML tags were added in the hip.conf file with their respective values. The settings for the new parameters were as follows:

```
<pkgparams>/usr/local/sbin/Certificates/pkgpubsig.bin</pkgparams>
```

```
<my_certificate>/usr/local/sbin/Certificates/CertChain</my_certificate>
<peer_certificate_required>yes</peer_certificate_required>
<send_certificates>yes</send_certificates>
```

8.4 Local Host Identity

There are two types of keys that must be stored for HIP; one is the own Host Identities which could be many in the system, including public and private key parts and the other is the host identities of peer hosts or systems. The files specified for those purposes are my_host_identities.xml and known_host_identities.xml.

my_host_identities.xml

The my_host_identities.xml file which is located at /usr/local/etc/hip/ contains public and private keys. For the security purpose, the my_host_identities.xml file have read and write permission only to the root. This protects the file from being compromised. However, the my_host_identities.xml file do not come with the installation. The identities file is created with the hitgen application. The command `./hitgen -type <RSA|DSA|IBE> -name <host_name> -bits <bits>` was used to generate the Host Identity and saved in the file called my_host_identities.xml in /usr/local/etc/hip/ directory. The type flag is used to choose the algorithm. [20] Currently the extended OpenHIP can generate RSA, DSA and IBE keys. However, the IBE identity key for identifying the host is yet to be deployed fully. The name flag takes the hostname of the machine. The bits flag is used to specify the length of the key in bits. The length of the key can be 512 bits, 1024 bits or 2048 bits. In case when two identities are required, at the end of the command –append is added. This command inserts the new identity at the end of the my_host_identities.xml file. Table 2 shows the syntax and lists the available options for the hitgen application.

Table 2. Syntax and available options for hitgen application. (Modified from [20])

Syntax
hitgen [-v] [-name <string>] [-type DSA|RSA|IBE] [-bits|length <NN>] [-anon] [-incoming] [-file <file>] [-publish] [-conf]
The optional parameters are: -v shows verbose debugging information -name <string> is the human-readable form of the HI -type sets the key type; DSA, RSA, IBE -bits specifies the length in bits for (P,G,Y) -length specifies the length in bytes for (P,G,Y) -anon sets the anonymous flag for this HI -incoming sets the allow incoming flag for this HI -file <file> write output to the specified file -publish extract HITs from the my_host_identities.xml file and create a file name <hostname>_host_identities.pub.xml -conf generate a sample conf file to usr/local/sbin/ directory (overwrites existing hip.conf file) -pkgfile generate the PKG public parameter file -noinput do not ask to seed random number generator

Following the syntax as shown in table 2 for this project, a 1024 bits length keys were generated using RSA algorithms in all hosts. The Host Identity for the host oh2 was generated with the following command.

```
root@oh2:/usr/local/sbin#./hitgen -type RSA -name oh2 -bits 1024
To seed the random number generator, please type some random text:
kdfjksle
Generating a 1024-bit RSA key
Generating RSA keys for HI...
Storing results to file 'my_host_identities.xml'.
```

Appendix 2 demonstrates the my_host_identities.xml file of the host oh2.

known_host_identities.xml

The known_host_identities.xml file is stored in the /usr/local/etc/hip/ directory. This file stores the known HITs of the peer hosts. With the installation default known_host_identities.xml is copied from the source. [20] The file contains tags like <name>, <HIT> and <LSI>. The name contains the hostname, HIT contains Host Identity Tag which is derived form the Host Identifier (HI) and LSI (Local Scoped Identifier) which is derived from the HIT. The IP address of the host identifier can be added using <addr> IP address </addr> tag. This is especially very helpful when the peer is not using the DHT or DNS. This tag will bind the IP address with the HIT. For instance the information of testing host was added in the oh2 host's known_host_identities.xml file as:

```
<host_identity alg="RSA" alg_id="3" length="128" anon="no" incoming=
"yes">
    <name>amitkc-1024</name>
    <addr>193.167.36.210</addr>
    <HIT>2001:1d:754d:50d9:b953:1ca4:d72e:fd16</HIT>
    <LSI>1.46.253.22</LSI>
</host_identity>
```

Appendix 3 shows the known_host_identities.xml file of the host oh2.

8.5 Extended HIP Daemon

After configuration, the application was started. The responder side starts the daemon by the `./hip -v -conf /usr/local/sbin/hip.conf` command. With this command the hip daemon initiates the TAP devices and starts listening to the port 139 for HIP messages. The oh2.hip.arcada.fi with Debian Linux acts as responder and the above command was applied. Similarly, the initiator starts the HIP daemon in the same way but with one additional parameter -t followed by the IP address of the responder. The Home PC acts as an initiator and the following command was executed.

```
$ ./hip -v -conf /usr/local/sbin/hip.conf -t 193.162.36.205.
```

The -v options is applied to view the verbose debugging messages whereas -conf options specifies the location of the configuration file, hip.conf. There are other debugging options and parameters available for starting the hip daemon. Table 3 shows the available options when starting the hip daemon.

Table 3. Available options for starting hip daemon.

```
root@oh2:/usr/local/sbin# ./hip –help
OpenHIP v0.7 daemon
Usage: hip [debug] [options]
Where debug is one of the following:
 -v         show verbose debugging information
 -q         quiet mode, only errors are shown
and options are:
 -d         daemon mode, fork and write output to logfile
 -r1        show pre-calculated R1 generation
 -o         opportunistic -- send NULL destination HIT in I1
 -a         allow any -- receive NULL destination HIT in I1
 -conf      <filename> absolute path to hip.conf file
 -p         permissive – does not enforce sigs, checksums (for
            debugging)
 -nr        no retransmit mode
 -t         <addr> manually trigger a HIP exchange with the given
            address
 -rvs       rendezvous server mode
With no options, simple output will be displayed.
```

As indicated by table 3, there are two debugging modes available and few other options. The -v option prints all the debugging messages in the terminal window whereas the -q debug option hides most of the debugging messages.

On starting the hip daemon, init_hip() and init_tap() functions are executed. The function init_tap() initializes the TAP device and turns the hip0 interface to up state. The location of the hip.conf file needs to be provided with the -conf option. To manually trigger the HIP Base Exchange with the responder the switch -t followed by the IP address of the responder is used. To start the hip daemon in the opportunistic

mode the switch -o is used. The opportunistic mode is used when the HIT of the intended responder is unknown.

After setting the user-provided daemon options, the daemon reads the size of the PKG public parameters created with the hitgen command. Then it reads the certificate chain from the location that has been specified in the configuration file and allocates the required memory for certificate payload. The absolute location of the certificate has to be specified inside the <my_certificate> tag of hip.conf file. Simultaneously other associated functions such as tunreader() thread, hip_esp_input() thread, hip_status() thread, hip_dns() thread, hip_pfkey() thread and hip_esp_output() thread are started. Eventually HIP threads initialization completes.

Furthermore, the new hip daemon reads the hip configuration file and the host ID file that is my_host_identities.xml file. Usually the configuration file is located in the /usr/local/sbin/ directory and the host ID file is located in the /usr/local/etc/hip/ directory in a Linux machine. The my_host_identities.xml may contain more than one identity based on same or different algorithm. The extended OpenHIP application can also generate a Host Identity based on an Identity Based Encryption (IBE) algorithm besides RSA and DSA. The DSA signature generation is faster than signature verification, whereas with the RSA algorithm, signature verification is very much faster than signature generation. Identity Based Encryption (IBE) is a ID-based cryptography which is a public-key encryption where the public-key is some unique information string about the identity of the user and it can be for e.g. user's email address, domain name or telephone/mobile number.

The hip daemon then reads the identities file and parses the host ID written in XML format. It checks if the HIT (Host Identity Tag) that is derived from the HI is valid or not. The reading and parsing the host identity and validating the HIT happen separately for each identity. After that HITs and LSIs of different identities are binded to the hip0 interface.

As soon as parsing the own host identities, the daemon reads and parses the host identity of known peers' from the known_host_identities.xml file. Afterwards a

precomputed R1 packet is generated. While generating the R1 packet, the information of certificate payload that can be fitted in one packet is also loaded.

In this project, we generated total of 7 certificates with different lengths. Two certificates were fitted in one packet and remaining certificates were sent in the UPDATE packet. Then the PF_KEY handler with usermode thread is registered and the HIP daemon starts listening to the sockets.

For measurement purpose, the extended OpenHIP uses gettimeofday() function for the timestamps on sending and receiving the HIP packets. The captured time is printed in the debugging messages. Appendix 4 shows condensed form of debugging messages.

When the initiator triggers the Base Exchange, the I1 packet is sent to the responder. I1 packet contains HIT of initiator and HIT of responder. The length of the I1 packet was 40 bytes.

When the responder receives the I1 packet on the socket, the HIP daemon selects one of the precomputed R1 packets and sends it to the initiator with the information of certificate payload appended to it.

The extended HIP daemon implements three phases for processing a R1 packet in the initiator side i.e. pass 0, pass 1 and pass 2. The pass 0 processes host ID, certificate and service offer, pass 1 processes the signature and pass 2 is separated for processing the other parameters.

The first phase (pass 0)

In the first phase, the daemon retrieves the host identity and signature. It checks the position, TLV (type-field value) and length of the HIP parameters R1_COUNTER, PUZZLE, DIFFIE_HELLMAN, HIP_TRANSFORM and HOST_ID. It determines the algorithm used in the public key. Then the HI in R1 packet validates the sender's HIT and verifies the R1 packet signature.

The second phase (pass 1)

In the second phase of processing, the R1 cookie is retrieved with the puzzle's difficulty (k) and its lifetime. Furthermore, it retrieves the random number (i) and checks for any opaque data that needs to be copied and send back as it is to verify that the same party has responded. Then the DH (Diffie-Hellman) public value is retrieved and it checks the Diffie-Hellman's prime and generator value and secret key set. Afterwards other R1 parameters HIP_TRANSFORM, HOST_ID, CERT, ESP_TRANSFORM, SERVICE and HIP_SIGNATURE_2 are checked for their position, type-field value and length.

The third phase (pass 2)

The main function in pass 2 is to compute the solution of the puzzle. The HIP daemon calculates and computes the solution (J) using the random number (I), HIT of initiator, HIT of receiver, puzzle difficulty (K) and SHA1 hash algorithm. The following formula is used to compute the solution for the puzzle.

$$Ltrunc(RHASH(I \mid HIT\text{-}I \mid HIT\text{-}R \mid J), K) == 0$$

The hip daemon solves the puzzle with brute force techniques by attempting repeatedly until the puzzle generates the matching solution (J). On solving the puzzle, the initiator sends the I2 packet with solution to the responder using the AES-CBC (Advanced Encryption Standard Cipher Block Chaining) algorithm for encryption and the HMAC-SHA1 algorithm for integrity. Besides solution, the I2 packet also contains certificates of the initiator.

When the responder receives the I2 packet, ESP_INFO, R1_COUNTER and SOLUTION are checked for length, and type-field values. The daemon retrieves the I2 cookie and decrypts the I2 data using the AES decryption key. Then the daemon checks the solution and signature of the I2 packet. If the signature and solution are correct, the responder immediately sends the R2 packet.

The Security Association (SA) is created with source and destination IP address with respective in and out SPIs. The Security Association policy is incremented with source

and destination HIT and finally the Base Exchange is completed between source and destination. In case of UPDATE packets, the Base Exchange is completed after sending and receiving all UPDATE packets. Appendix 5 shows HIP parameters that have been used in HIP packet in Base Exchange.

8.6 Generating Certificates

The standard x.509 certificates were generated with the latest version of openSSL application (OpenSSL-1.0.0). A certificate chain with seven certificates was created. In this project, N certificate levels were generated signed. The first level certificate is signed by the root certificate. The second level certificate is signed by the certificate from the first level and so on henceforth. First a self signed root certificate was generated and then the first level certificate was generated signed by the root certificate. Afterwards, the second level certificate was generated signed by the first level certificate. For the calculation of Base Exchange time in different scenarios, six levels of certificates were created. Each certificate was signed by the certificate from previous level. Appendix 6 shows the snapshot windows while generating certificates in the host oh2.hip.arcada.fi host.

9 Performance Measurements

As the goal of the project was to measure the time complexity of Base Exchange for the extended OpenHIP, several experiments were done from the local and remote network. The time consumption of Base Exchange without certificates and with different number of certificates was measured in the original OpenHIP v0.7 and the extended OpenHIP respectively. The main reason behind the performance measurements was to find out how much of overhead time would be added to the HIP Base Exchange in extended OpenHIP when certificates are used and to compare it to the original OpenHIP's Base Exchange.

9.1 Experiment 1

The first set of experiments was done with the extended OpenHIP in the local IP network. The initiator was a host with IP address 193.167.36.210 and the responder was oh2.hip.arcada.fi host with IP address 193.167.36.205. First of all the Base Exchange was performed without any certificates. Secondly, the experiment was done with 2 certificates. As the length of each certificate was 653 bytes they both were fitted into one packet. The responder's certificates were delivered in R2 packet whereas the initiator certificates were delivered in I2 packet. In the third experiment, 4 certificates were delivered and in this experiment one extra UPDATE packet was required as maximum of two certificates were fitted into R2 packet. The remaining certificates had to be delivered in an UPDATE packet. In the same way, two more certificates were added in the next experiment and one more certificates in the next experiment. To deliver all seven certificates, three UPDATE packets were required. Altogether five sets of experiment were performed and each set of experiment was done 10 times for strong approximation in the measurements. Table 4 shows the calculated results of the measurements. The times are in seconds.

Table 4. Measurement results from Experiment 1.

			AVERAGE	STDEV	MIN	MAX	RANGE
1	without cert	I	0.078342032	0.004311156	0.072589874	0.087229967	0.014640093
		R	0.074630976	0.004272395	0.068920135	0.083770037	0.014849901
2	2 cert	I	0.068969481	0.008729394	0.047867	0.080960035	0.033093035
		R	0.064699735	0.007127785	0.046519995	0.072770119	0.026250124
3	4 cert	I	0.075799095	0.003774055	0.071536	0.08427	0.012734
		R	0.072533394	0.003919844	0.068160057	0.080450058	0.012290001
4	5 cert	I	0.075521101	0.007974844	0.05471015	0.083549976	0.028839827
		R	0.071531373	0.008109422	0.051640034	0.080549955	0.028909922
5	7 cert	I	0.081896996	0.005237	0.073969841	0.089020014	0.015050173
		R	0.078293037	0.005364	0.07103014	0.085600138	0.014569998

As indicated in table 4, there were five scenarios from the Base Exchange without certificates to the Base Exchange with seven certificates. In the third column, I refers to the initiator and R refers to the responder. The fourth column shows the average time consumption by initiator and responder to complete the Base Exchange. The fifth column shows the standard deviation. The sixth and seventh column show the minimum and maximum time consumed to complete the Base Exchange among the 10 samples. The eighth column shows the range. The standard deviation depicts the average minimum and maximum deviation from the mean time whereas the range shows the difference between minimum and maximum time. The graphical representation of average time taken to complete the Base Exchange in different cases is shown in Figure 11.

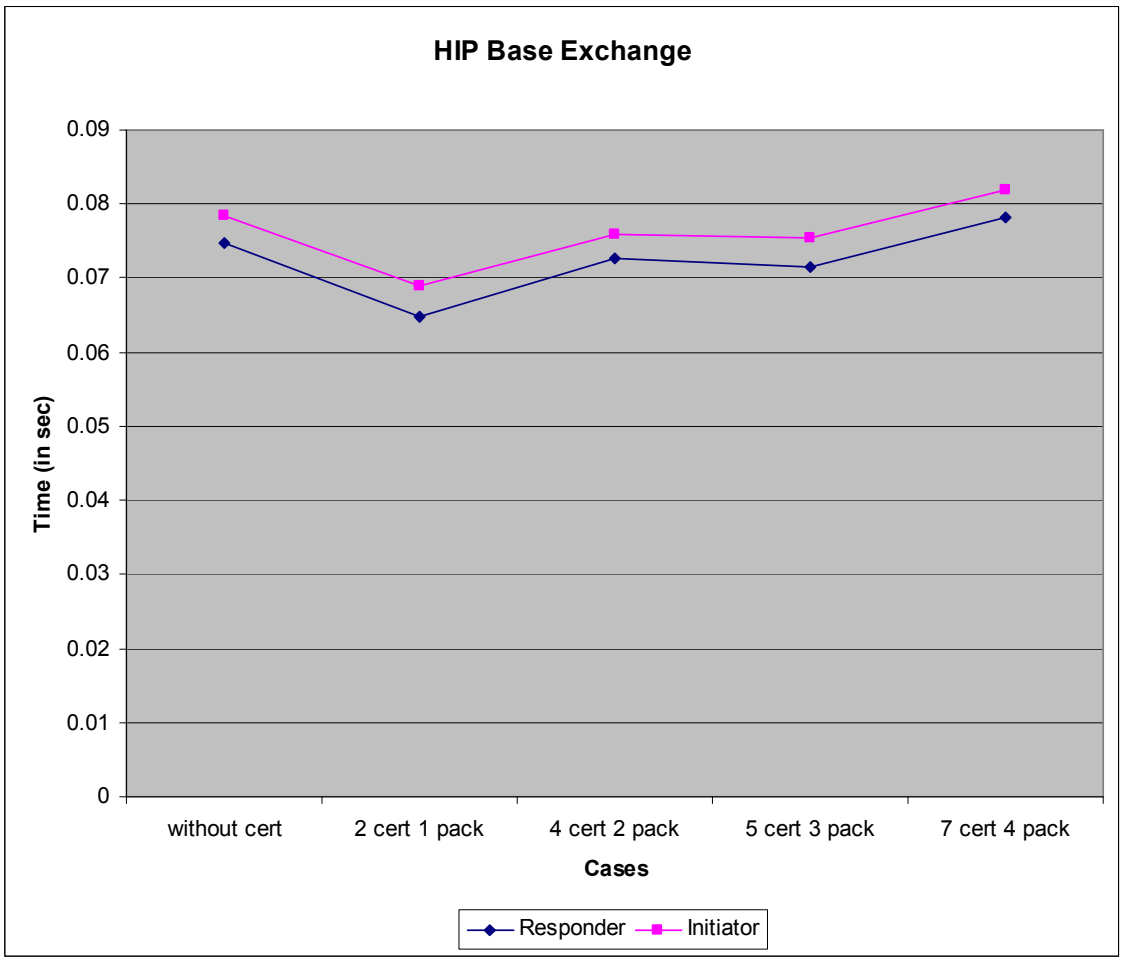

Figure 11. Graphical representation of average time consumption for Base Exchange in experiment 1.

As shown in figure 11, with the increasing number of certificates the time of Base Exchange also increases. The x-axis in figure 11 shows five cases with and without certificates. The y-axis shows the time consumption in seconds. The blue line indicates the responder and the pink line indicates the initiator. However the Base Exchange time without certificates is significantly higher than the Base Exchange with two certificates.

9.2 Experiment 2

In the same way, another set of experiments was done from a remote computer. In this experiment, a home PC was used as an initiator and the responder was the same host, oh2.hip.arcada.fi in the domain hip.arcada.fi. The home PC was connected to

Sonera's gateway and the host oh2 was connected to the Funet WAN through gateways in arcada.fi. The experiment was done in the same order as experiment 1. The initiator sent two certificates and the responder sent 2, 4, 5 and 7 certificates depending on the scenario. The measurements can be seen in Table 5.

Table 5. Measurement results from Experiment 2.

			AVERAGE	STDEV	MAX	MIN	RANGE
2	without cert	I	0.072052979	0.005278484	0.080780029	0.062910080	0.017869949
		R	0.067508960	0.004887479	0.077859879	0.059989930	0.017869949
3	2 cert	I	0.067731977	0.006319700	0.078999996	0.058990002	0.020009995
		R	0.063967967	0.005829584	0.074869871	0.055739880	0.019129992
4	4 cert	I	0.072272015	0.005766193	0.079240084	0.062540054	0.016700029
		R	0.068173003	0.005082995	0.074490070	0.059309959	0.015180111
5	5 cert	I	0.074768113	0.005941499	0.083080053	0.065720081	0.017359972
		R	0.071485888	0.005760826	0.079689980	0.062651000	0.017038980
6	7 cert	I	0.073205948	0.003249558	0.079649925	0.069759846	0.009890079
		R	0.069992018	0.003306937	0.076659918	0.066680193	0.009979725

Table 5 shows the time averages of ten Base Exchanges when Responder is sending zero, two, five and seven certificates. Four certificates were sent in R2 and in one UPDATE packet. Five certificates and seven certificates demanded two and three UPDATE packets respectively. In between the two hosts there were 8 hop counts. Again, the times are in seconds. Graphically the average time taken in experiment 2 to complete the Base Exchange in different scenarios is illustrated in figure 12

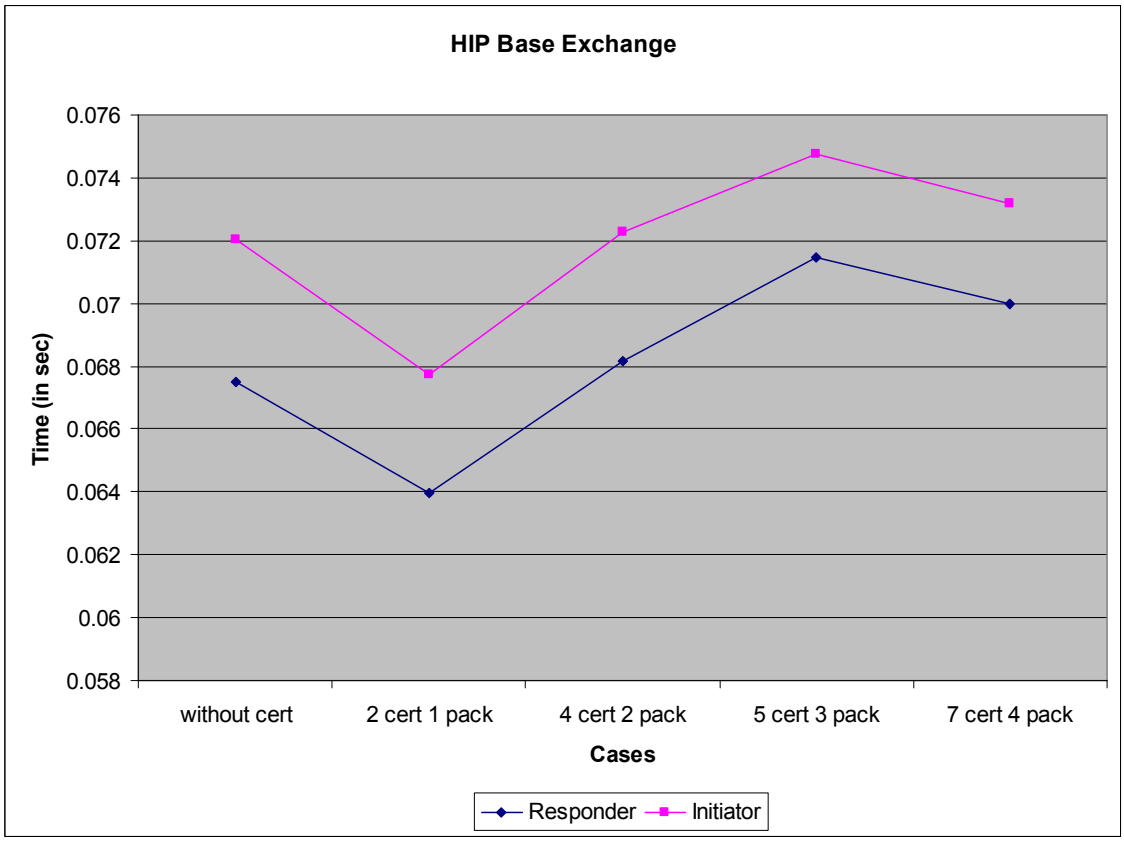

Figure 12. Graphical representation of average time consumption for Base Exchange in experiment 2.

As shown by figure 12, the Base Exchange took little bit more time from the remote host than from a host in the local IP network. Here again, the x-axis in figure 11 shows five cases with and without certificates. The y-axis shows the time consumption in seconds. The blue line indicates the responder and the pink line indicates the initiator.

The measurements taken with extended OpenHIP had to be compared with unmodified OpenHIP though only one scenario, that is, without certificates. That is why one more set of experiment was done with the unmodified OpenHIP v0.7. In the same manner, ten experiments were done; collected data and times were calculated. The measurements gave the results of average 0.061676049 seconds ≈ 62 milliseconds for initiator and average 0.061790016 seconds ≈ 62 milliseconds for responder in the remote network. From the local host, the average time taken to complete the Base Exchange by the initiator was 0.054104995 seconds ≈ 54 milliseconds and by the responder was 0.054187348 seconds ≈ 54 milliseconds.

The Base Exchange consists of five types of packet namely I1, R1, I2, R2 and UPDATE. The average processing time for individual packets differs from each other. The R1 packet consumes comparatively more time for processing than other HIP packets. Moreover, the processing time of I2 packets also depends on the difficulty level of the puzzle. The default difficulty level was 10. Table 6 shows the processing time for individual packets in unmodified and modified OpenHIP with the default difficulty level.

Table 6. Average time for processing individual packets.

			I1	**R1 with certs**	**R1 without certs**	**I2**	**R2**	**UPDATE**
1.	original OpenHIP v0.7	I	-	-	0.036314988	0.024878025*	0.000401998	-
		R	0.000211024	-	0.03860302**	0.022551966	0.00031004***	-
2.	Extended OpenHIP	I	0.003667998*	0.033060002	0.040844989	0.026989985*	0.003296041	0.003205967
		R	0.000829458	0.057582045**	0.064041996**	0.002875996	0.003683996***	0.003187943

Note:

* includes network time for sending I1/I2 + processing time of I1/I2 by responder + network time for receiving R1/R2 by the initiator

** includes network time for sending R1 + processing time of R1 by initiator + network time for receiving I2 by the responder

*** includes network time for sending R2 + processing time of R2 by initiator + network time for receiving SA complete message by the responder

In table 6 the measurements are again in seconds. The times in I1 and I2 include the transmission delay for sending I1/I2 and similarly, the times in R1 and R2 also include the transmission delays. The transmission delays varied depending on the network traffic.

The measurements show that implementing certificates does not consume much time in the Base Exchange. When sending two certificates, the Base Exchange took 2 milliseconds more time for initiator and 6 milliseconds more time for responder in comparison with the Base Exchange of unmodified OpenHIP v0.7 without certificates. With seven certificates, the differences were 8 milliseconds and 11 milliseconds respectively.

10 Discussion

One of the main intentions in carrying out this project was to modify and extend the existing OpenHIP software for handling certificates and thus establish trust between the network hosts. Besides learning and extending the knowledge of cryptography and knowledge about Host Identity Protocol were great benefits.

While carrying out this project, developing the infrastructure was not hard but meeting all the dependencies for installation was a little bit challenging. Though there were just four dependencies in general to be fulfilled for extended OpenHIP, those four dependencies need other dependencies. As a result, a chain of dependencies were created to be installed to complete all requirements. Among different versions of dependencies available, only a particular version, that is libpbc-dev 0.5.10, worked perfectly with the application. Furthermore, finding packages for Debian distribution with appropriate architecture was also difficult and for this reason dependencies were mostly compiled from the source. A number of compatibility and dependability issues and problems were faced in this project and took some ample time to be fixed. Despite all these troubles, the extended OpenHIP thrived and was tested successfully in the designed infrastructure.

Owing to the time constraint, I could not cover the whole project. The project is still work in progress state and in the phase of implementing IBE based public keys to establish trusted relationship between the peers. In the same way, analysis on HIP multicast models when certificates are used is also in progress.

Regarding the future work, the extended application can be further developed for NAT traversal so that it can handle certificates with the hosts that are beyond a NAT. Similarly using certificates to support mobility and multihoming scenario is another extension that can be implemented. As the current version of extended OpenHIP only supports Debian distributions of Linux, it can be developed to support other popular distributions such as Ubuntu, Centos, Fedora, etc. and also Microsoft Windows versions. However, this thesis can be used as a reference when augmenting the project.

11 Conclusion

The project concentrated on studying the possibilities to establish trusted hosts in Host Identity Protocol and implement certificate handling in HIP packets. Additionally, to measure the time complexity in Base Exchange while handling the certificates was also part of the project. DNSSEC gave trust only to the initiator while certificate based authentication of public keys gave trust both to initiators and responders. For the implementation, open source software developed by Boeing Company was adopted. Several modifications were made and extensions were added to handle the certificates and verifying them. Likewise for the measuring the time complexity in Base Exchange when HIP control packets are accompanied with certificates, several experiments were performed locally and remotely and results were calculated. The project implemented standard x.509 certification of the public keys used as HIs to acquire the trust between the hosts.

The results of the project show that the x.509 certificates can be successfully implemented to derive trust and it is not so complicated from the user point of view. Several important functions were added, mainly in hip_input.c and hip_output.c source files for handling the certificates and service offers. Moreover, a few XML tags were added and configured to be used in the configuration file. Similarly, the results from the measurements show that the encapsulation of certificates does not significantly affect the Base Exchange time. Few milliseconds were consumed more in comparison with the unmodified OpenHIP v0.7. Finally, the extended version of OpenHIP with certification support was developed and tested successfully and the goal of the project was achieved. However, there is always room for improvements; further enhancement projects can be conducted.

References

1 Gurtov A. Host Identity Protocol (HIP) Towards the Secure Mobile Internet. United Kingdom: John Wiley & Sons Ltd; 2008.

2 Moskowitz R, Nikander P. RFC 4423 Host Identity Protocol (HIP) Architecture. The Internet Engineering Task Force (IETF); 2006.

3 Aura T, Nagarajan A, Gurtov A. Analysis of the HIP Base Exchange Protocol [online].
URL: http://hipl.infrahip.net/papers/analysis_hip.pdf. Accessed 4 May 2012.

4 Moskowitz R, Nikander P, Jokela P, Henderson T. RFC 5201 Host Identity Protocol. The Internet Engineering Task Force (IETF); 2008.

5 Al-Shraideh F. Host Identity Protocol [online].
URL:http://www.netlab.tkk.fi/opetus/s38030/k05/HostIdentityProtocol.doc.
Accessed 4 May 2012.

6 Mobile and Wireless Communication Systems. Arcada University of Applied Sciences [online].
URL: http://wireless.arcada.fi/MOBWI/material/HM_1_2.html. Accessed 4 May 2012.

7 HIP for Linux. Helsinki Institute for Information Technology [online].
URL: http://infrahip.hiit.fi/index.php?index=how. Accessed 4 May 2012.

8 Mattsson J. Mobile Data Communication based on Host Identity Protocol (HIP). Bachelor of Science thesis. Helsinki: Arcada University of Applied Sciences; 2010.

9 Boyd C, Nieto J. Information Security and Privacy. 10th Australasian Conference. Gremany: Springer-Verlag; 2005.

10 Khurri A, Vorobyeva E, Gurtov A. Performance of Host Identity Protocol on Lightweight Hardware. Helsinki: Helsinki University.

11 Bergström L, Fröjdman J, Grahn K, Karlsson J, Pulkkis G. Host Identity Protocol (HIP) as a Virtual Learing Object. Proceedings of Informing Science+IT Education Conference. Varna, Bulgaria; June 22-25 2008.

12 Lagarier J, Eggert L. RFC 5204 Host Identity Protocol (HIP) Rendezvous Extension. The Internet Engineering Task Force (IETF); 2008.

13 Gomez M. Rendezvous Mechanisms in Host Identity Protocol. Diploma thesis. Universitat Polit`ecnica de Catalunya; 2006.

14 Zhou B. Opportunistic Security of Host Identity Protocol. Master's thesis. Helsinki University of Technology; 2006.

15 Timo Karvi. Computer Security, Part IV: Host Identity Protocol [online]. Helsinki: Helsinki University; 2010.

16 Heer T, Varjonen S. Host Identity Protocol Certificates. The Internet Engineering Task Force (IETF); 2011.

17 Forsgren H, Grahn K, Karvi T, Pulkkis G. Security and Trust of Public Key Cryptography Options for HIP. IEEE International Conference on Computer and Information Technology; 2010.

18 Forsgren H, Grahn K, Karvi T, Pulkkis G. Public Key Cryptography Options for Trusted Host Identities in HIP. WISEciti Public Seminar; 2010.

19 Pellika J. HIP Certificate Requests. The Internet Engineering Task Force (IETF); 2011.

20 Boeing Company. OpenHIP [online]. MediaWiki; 23 March 2012.
URL: http://www.openhip.org/wiki/index.php?title=Overview. Accessed 16 May 2012.

21 Fröjdman J. Installation and evaluation of a HIP infrastructure. BSc Thesis. Helsinki: Arcada University of Applied Sciences; 2008.

22 HIP for Linux [online]. Helsinki Institute for Information Technology; 2004
URL: http://infrahip.hiit.fi/index.php?index=about. Accessed 16 May 2012.

23 HIP for inter.net Project [online]. Ericsson Ab, NomadicLab; 2008.
URL: http://hip4inter.net/. Accessed 16 May 2012.

24 Kuptsov D. Implementing CuteHIP: Feasibility Analysis of Java-based Network-layer Security Protocols. Finland: Aalto University.

25 The Pairing-Based Cryptography Library [online].
URL: http://crypto.stanford.edu/pbc. Accessed 4 May 2012.

26 The GNU Multiple Precision Arithmetic Library [online].
URL: http://gmplib.org. Accessed 4 May 2012.

Appendices

Appendix 1. Final configuration file, hip.conf, of the host oh2.

```
root@oh2:/usr/local/sbin# cat hip.conf
<?xml version="1.0" encoding="UTF-8"?>
<hip_configuration>
  <cookie_difficulty>10</cookie_difficulty>
  <packet_timeout>10</packet_timeout>
  <max_retries>5</max_retries>
  <sa_lifetime>900</sa_lifetime>
  <loc_lifetime>1800</loc_lifetime>
  <preferred_hi>default-1024</preferred_hi>
  <send_hi_name>yes</send_hi_name>
  <dh_group>3</dh_group>
  <dh_lifetime>900</dh_lifetime>
  <r1_lifetime>300</r1_lifetime>
  <failure_timeout>50</failure_timeout>
  <msl>5</msl>
  <ual>600</ual>
  <hip_sa>
    <transforms>
      <id>1</id>
      <id>2</id>
      <id>3</id>
      <id>4</id>
      <id>5</id>
      <id>6</id>
    </transforms>
  </hip_sa>
  <esp_sa>
    <transforms>
      <id>1</id>
      <id>2</id>
      <id>3</id>
      <id>4</id>
      <id>5</id>
      <id>6</id>
    </transforms>
  </esp_sa>
  <!--<dht_server>192.168.0.2</dht_server>
  <dht_server_port>5851</dht_server_port>-->
  <!--<dns_server>192.168.0.2</dns_server>-->
  <disable_dns_lookups>yes</disable_dns_lookups>
  <disable_notify>no</disable_notify>
  <disable_dns_thread>no</disable_dns_thread>
  <enable_broadcast>no</enable_broadcast>
  <!--<preferred>192.168.0.1</preferred>-->
  <!--<preferred_interface>eth0</preferred_interface>-->
  <save_known_identities>no</save_known_identities>
  <send_certificates>yes</send_certificates>
  <peer_certificate_required>yes</peer_certificate_required>
  <cert_service_offer>send</cert_service_offer>
  <pkgparams>/usr/local/sbin/Certificates/pkgparamsig.bin</pkgparams>
 <my_certificate>/usr/local/sbin/Certificates/CertChain</my_certificate>
</hip_configuration>
```

Appendix 2. Host Identity file, my_host_identities.xml, of testing host.

```
root@amitkc:/home/amit# cat /usr/local/etc/hip/my_host_identities.xml
<?xml version="1.0" encoding="UTF-8"?>
<my_host_identities>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes" r1count="10">
    <name>amitkc-1024</name>
<N>CFBC127ECE9644654E0F05A67E822E7617B2B65D0B3ADA860420E56E91BF5266557D5F
0618904C87C609995F4C11FBFDF15BC87A6248620F10991C5A6A6C76ECB2FF7FF413710B9
AE999C8064E2E6A061181D92B65501D040BD7B332B8881A6B3949838F969F9173376A480F
6081E2A5F273D2BD76FB7614FCECE8EC303D66E9</N>
    <E>010001</E>
<D>88ACF602468BE97E39A87A903BA60D499777F6B22B0452E85FE900EB09ABB65C2C1D2B
9B082850A1F833329CD7A0F189F7F5EA665329CA929CDA102490E6C22AF46C266FFE86C6F
44850171E8576DA4CBDA9528793BCBB59BED26D48E61D44DC556CBAD78303E86196FD51F0
91E9D9B048020E395A2DC367F77230173E7390B1</D>
<P>EB02D4C18339914BED0A417A1825EA1E199DAB93086186C678A5F414955D267156DC56
DB8EA0D53EF1955B9CB45948A67F4558A601195BAFF306F49E84868295</P>
<Q>E2499B9FAE457E07DD6E313F1141C0888DD962268DD0DD8DBAEC913BF4F3E61A834A69
5C5900F40A0D5A60924E56B60B31FA915E511EC9C9F21E92A5EA7CF205</Q>
<dmp1>B7500E04A407537E95F759FB92BA8053DA47527DEE1BA4B54B86A74D26F4F1F7967
96886FA4A1A0C6F04C1E10A0C5ACF049762FE99FCDDB7F1298BC5DA39D1C1</dmp1>
<dmq1>53C77CDEB9DEF0B268C10B6A963109DF84E51EF9B737C54F5C0D8A17F0B0FC58849
3603ED89509C54EFC4F14DBFC2E9A267EE6A34CD561506B5BF65F87598B21</dmq1>
<iqmp>B5CFEE457DBDFA7BE237684031900A528DBA55DA050635013D20CBF6BAE1AA13B93
80D9F4CAEDFEAF7EF561FB7EE99A221744E8AC9D2A100E68496FD97861175</iqmp>
    <HIT> 2001:15:4dc9:4d8d:2cd8:e61:8089:af99</HIT>
    <LSI> 1.137.175.153</LSI>
  </host_identity>
  <host_identity alg="IBE" alg_id="7" length="128" anon="no"
incoming="yes" r1count="10">
    <name>amitkc-1024</name>
<P>080502e925c7bdca1500b8f8cf1896db6d5df2b4010b33669c88c0dc99007e3dffd64f
2f9f00e4e2e99aaacbf2184b8fad69ce29b68d02fcbd968a7a5f11db04f7852ef07a0ccd4
7679ff6a8edb0025561944245e92f12a57fc1bc58615bb089ddb20527d863ea6b6aa9438e
a2ecb0f37be5dc27bb28c7e957c8b3073cbf8ac900</P>
<Ppub>080502e925c7bdca1500b8f8cf1896db6d5df2b4010b33669c88c0dc99007e3dffd
64f2f9f00e4e2e99aaacbf2184b8fad69ce29b68d02fcbd968a7a5f11db04f7852ef07a0c
cd47679ff6a8edb0025561944245e92f12a57fc1bc58615bb089ddb20527d863ea6b6aa94
38ea2ecb0f37be5dc27bb28c7e957c8b3073cbf8ac900</Ppub>
    <HIT>2001:14:af96:fe3d:1ef0:7e9f:c654:399e</HIT>
<K>02c1f621d7be692855ce5c404ad6e1a348bd506298eab3e14fe6ac2758db106ba41a32
0e819dc3f7ec59986e147c3af9c3bbfbecf97f8398f67368e463c85402efdb3714644cb7e
a7bc5c11b9e5cc9a270b2f75fa0873bfa672b299940c23dde7d86bc96592f0dbad40af4c8
693e5d43280c058a57b17d2cef163ce9f12a1a6000</K>
    <LSI>1.84.57.158</LSI>
  </host_identity>
</my_host_identities>
```

Appendix 3. Known peers' identity file, known_host_identities.xml, of the host oh2.

```
root@oh2:/usr/local/etc/hip# cat known_host_identities.xml
<?xml version="1.0" encoding="UTF-8"?>
<known_host_identities>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes">
    <name>hipserver.mct.phantomworks.org-1024</name>
    <HIT>2001:14:4dcd:2a09:74a:caee:2a0:ec4a</HIT>
    <LSI>1.230.120.200</LSI>
  </host_identity>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes">
    <name>crossroads.infrahip.net-1024</name>
    <addr>193.167.187.134</addr>
    <HIT>2001:19:b673:8406:e32d:6754:db0b:cde7</HIT>
    <LSI>1.11.205.231</LSI>
  </host_identity>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes">
    <name>woodstock4.hip4inter.net-1024</name>
    <addr>193.234.218.202</addr>
    <HIT>2001:15:1522:ed39:3a8f:5ef2:eb5:cc30</HIT>
    <LSI>1.181.204.48</LSI>
  </host_identity>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes" r1count="21">
    <name>amitkc-1024</name>
    <addr>80.221.27.14</addr>
    <HIT>2001:15:4dc9:4d8d:2cd8:e61:8089:af99</HIT>
    <LSI>1.137.175.153</LSI>
  </host_identity>
  <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes" r1count="10">
    <name>oh2-1024</name>
    <addr>193.167.36.205</addr>
    <HIT> 2001:12:397f:d41:aeab:6e33:9071:f496</HIT>
    <LSI> 1.113.244.150</LSI>
  </host_identity>
 <host_identity alg="RSA" alg_id="5" length="128" anon="no"
incoming="yes" r1count="10">
    <name>oh1-1024</name>
    <addr>193.167.36.204</addr>
    <HIT>2001:1a:3d9c:266:2142:3d3c:ca06:4186</HIT>
    <LSI>1.6.65.134</LSI>
  </host_identity>
</known_host_identities>
```

Appendix 4. Initiator debugging messages of the host oh2.

```
root@amitkc:/usr/local/sbin# ./hip -v -conf hip.conf -t 193.167.36.204
init_hip()
init_tap()
Using TAP device hip0.
>>>>SKIPPED>>>>>>>
Total certificate payload length is 2112 bytes.
>>>>SKIPPED>>>>>>>
My host identities:
 HI: amitkc-1024 HIT: 2001:15:4dc9:4d8d:2cd8:e61:8089:af99
    LSI: 1.137.175.153
>>>>SKIPPED>>>>>>>
Known peer host identities:
 HI: oh2-1024 HIT: 2001:12:397f:d41:aeab:6e33:9071:f496
    LSI: 1.113.244.150 [193.167.36.205]
 HI: oh1-1024 HIT: 2001:1a:3d9c:266:2142:3d3c:ca06:4186
    LSI: 1.6.65.134 [193.167.36.204]
>>>>SKIPPED>>>>>>>
hip_certs_len: total of 2 certificates will fit into a packet.
R1 packet has 1336 byte certificate payload. (location = 1816)
>>>>SKIPPED>>>>>>>
Thu May 10 13:50:09 2012 (1) Sending HIP_I1 packet (40 bytes)...
>>>>SKIPPED>>>>>>>
Thu May 10 13:50:09 2012 (1) Received HIP_R1 packet from 80.221.27.14 on
raw socket length 1964
>>>>SKIPPED>>>>>>>
R1 pass 0.
 R1 pos = 40 TLV type = 128 (PARAM_R1_COUNTER) length = 12
 R1 pos = 56 TLV type = 257 (PARAM_PUZZLE) length = 12
 R1 pos = 72 TLV type = 513 (PARAM_DIFFIE_HELLMAN) length = 195
 R1 pos = 272 TLV type = 577 (PARAM_HIP_TRANSFORM) length = 12
 R1 pos = 288 TLV type = 705 (PARAM_HOST_ID) length = 148
Found RSA HI with public modulus: 0x be691d5d ce595363 adbaed47 f4669e5a
f84bbeea c38dd644 97cc292d 5151571a
>>>>SKIPPED>>>>>>>
HI has name: amitkc-1024 length: 8
HI in R1 validates the sender's HIT.
Can verify R1 packet signature.
R1 pass 1.
>>>>SKIPPED>>>>>>>
Got the R1 cookie: (k=15 lifetime=39 (128 seconds) opaque=0
I=0xef971524bfa105d4)
>>>>SKIPPED>>>>>>>
Got DH public value of len 192: 0x d8538d4a ab8ad866 06c63eec 0d414155
d85522d5 d7b6ba1b cc694382 c21a37cc
Diffie-Hellman-Parameters: (1536 bit)
    prime:
        00:ff:ff:ff:ff:ff:ff:ff:c9:0f:da:a2:21:68:
>>>>SKIPPED>>>>>>>
    generator: 2 (0x2)
************
DH secret key set to:
0x 821e894e a219f9db 093ae231 262a8410
>>>>SKIPPED>>>>>>>
 R1 pos = 440 TLV type = 768 (PARAM_CERT) length = 657
 R1 pos = 1104 TLV type = 768 (PARAM_CERT) length = 65
```

```
R1 pass 2.
>>>>SKIPPED>>>>>>>
Calculating Ltrunc(SHA1(I|Rand),K)...found match in 5116 tries (~0
seconds).
MD= 0ca7760f 30f0bf04 2e6ab5b9 d2174204 bbc88000
IJ= d405a1bf 241597ef 20010012 397f0d41
aeab6e33 9071f496 2001001a 3d9c0266
>>>>SKIPPED>>>>>>>
Sending the I2 cookie: (k=15 lifetime=39 (128 seconds) opaque=0
I=0xef971524bfa105d4)
solution: 0xaea4dd10eeed4552
Using HIP transform of 1.
Drawing new HIP encryption/integrity keys:
Key 0 (1,16) keymat[  0] 0x bd7ed594 72e4d3e3 3fce0cbb 630338da
>>>>SKIPPED>>>>>>>
AES encryption key: 0x e8b329dc 49c4d54a 75543ce2 5095c10b
Encrypting 160 bytes using AES.
 SHA1:  75b67b45 d321ca30 2e266f11 97f48071 bb65de64
Signature:  431d5955 03b975ca ba3581b5 0d61ee47
58c5a757 0ce05934 f1086c04 8e03309c
>>>>SKIPPED>>>>>>>
Thu May 10 13:50:09 2012 (1) Sending HIP_I2 packet (2008 bytes)...
>>>>SKIPPED>>>>>>>
Read 1564 bytes from socket.
Thu May 10 13:50:09 2012 (1) Received HIP_R2 packet from 80.221.27.14 on
raw socket length 1564
>>>>SKIPPED>>>>>>>
Found 653 bytes of certificate payload.
Found 653 bytes of certificate payload.
>>>>SKIPPED>>>>>>>
HMAC_2 verify over 1536 bytes. hdr length=191
HMAC_2 verified OK.
SHA1:  dc1fa821 e4165b68 e9c2b74d 06973752 b26c56ac
RSA HIP signature is good.
>>>>SKIPPED>>>>>>>
Thu May 10 13:50:09 2012 (1) Received UPDATE packet from 80.221.27.14 on
raw socket length 1556
>>>>SKIPPED>>>>>>>
HIP exchange complete.
Thu May 10 13:50:09 2012 (1) Adding security association:
    src ip = 80.221.27.14 dst ip = 193.167.36.205
    SPIs in = 0x8c9a69c7 out = 0x0
Thu May 10 13:50:09 2012 (1) sadb_add(src=80.221.27.14,
dst=193.167.36.205, src HIT=2001:15:4dc9:4d8d:2cd8:e61:8089:af99, dst
HIT=2001:12:397f:d41:aeab:6e33:9071:f496, spi=0x0, direction=out)
spi=0x0 ekey: 0x 65436b59 2a348022 154fde43 c1e4f4f0
spi=0x0 akey: 0x f6306306 91e01e91 20a8d1a9 e7e85c74 1fcd4188
hitMagic checksum over 32 bytes: 0xe3d2
Thu May 10 13:50:09 2012 (4) Base exchange completed from
    2001:15:4dc9:4d8d:2cd8:e61:8089:af99 / 80.221.27.14 to
    2001:12:397f:d41:aeab:6e33:9071:f496 / 193.167.36.205
```

Appendix 5. HIP parameters.

Table 7. HIP parameters used in the project. (Modified from [4])

Parameters	Type	Length	Data
R1_COUNTER	128	12	System boot counter
PUZZLE	257	12	K and random #I
SOLUTION	321	20	K, Random #I and puzzle solution J
DIFFIE_HELLMAN	513	variable	public key
HIP_TRANSFORM	577	variable	HIP Encryption and Integrity Transform
HOST_ID	705	variable	Host Identity with Fully-Qualified Domain FQDN (Name) or Network Access Identifier (NAI)
CERT	768	variable	HI Certificate; used to transfer certificates.
NOTIFICATION	832	variable	Informational data
ECHO_REQUEST_SIGNED	897	variable	Opaque data to be echoed back; under signature
ECHO_RESPONSE_SIGNED	961	variable	Opaque date echoed back; under signature
HMAC	61505	variable	HMAC-based message authentication code with key material from HIP_TRANSFORM
HMAC_2	61569	variable	HMAC-based message authentication code with key material from HIP_TRANSFORM. Compared to HMAC, the HOST_ID parameter is included in HMAC_2 calculation
HIP_SIGNATURE_2	61633	variable	Signature of the R1 packet
HIP_SIGNATURE	61697	variable	Signature of the packet
SERVICE	65334	variable	Service offer
SERVICE ACK	65334	160 bit	Service offer acknowledgment

Table 7 shows different types of HIP parameters used in this thesis work with respective type, length and data in it.

Appendix 6. Generating certificates in the testing host.

Figure 13 shows how the RSA private key and public key parameters file was generated in this thesis work.

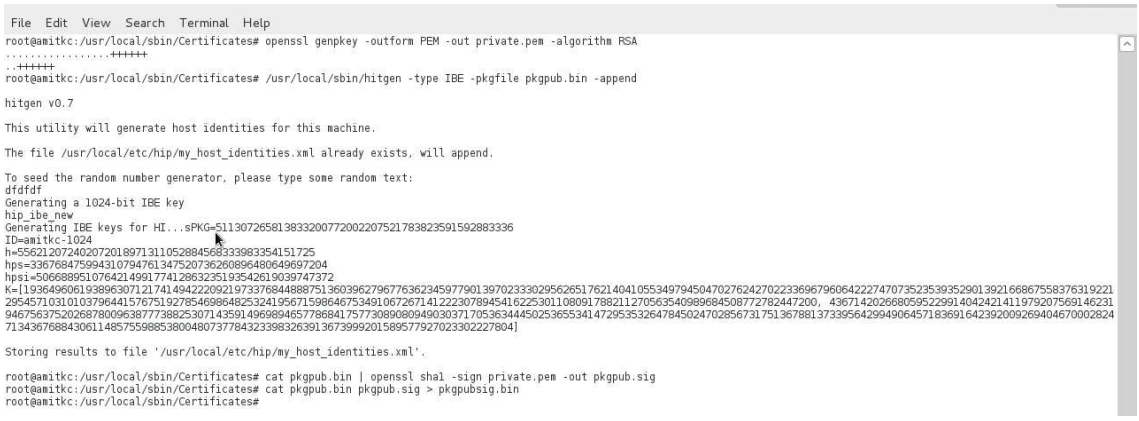

Figure 13. RSA private key and public key parameter generation.

As shown in figure 13, first the RSA private key, private.pem, was generated using OpenSSL and then hitgen utility was used to create public parameter file, pkgpub.bin. After that signed SHA1 signature of pkgpub.bin was calculated with private.pem and finally pkgpub.bin and signed signature file was combined together resulting in a file pkgpubsig.bin which was used in the HIP configuration file for handling certificates.

Figure 14 shows how self-signed root certificate was created.

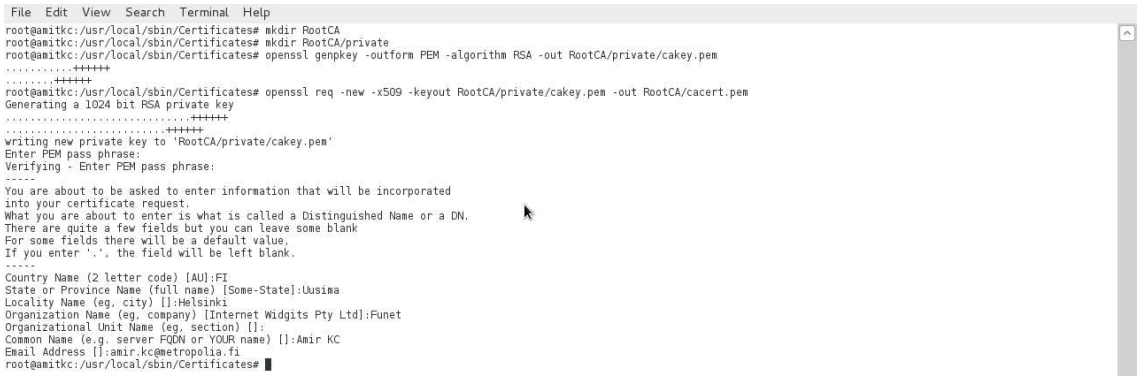

Figure 14. Generation of self-signed root certificate.

As illustrated by figure 14, a self-signed root certificate was generated with OpenSSL providing necessary informations such as country name, state, email address, etc.

Figure 15 shows how certificate chains were created in this thesis work.

Figure 15. Creating certificate chains

As shown in figure 15, certificate chain was created for the sixth level and saving in the Level6CA directory. Firstly, RSA private key, cakey.pem, was generated in the sixth level and then certificate chain file, cacert.pem, was created signed by the cakey.pem in the fifth level providing necessary x509 certificate parameters. Other certificate chains were created applying the same methods.